(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
15 March 2001 (15.03.2001)

PCT

(10) International Publication Number
WO 01/018653 A1

(51) International Patent Classification[7]: G06F 12/10

(21) International Application Number: PCT/US00/24078

(22) International Filing Date:
1 September 2000 (01.09.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/152,680 7 September 1999 (07.09.1999) US

(71) Applicant: **FAST-CHIP INCORPORATED** [US/US]; 950 Kifer Road, Sunnyvale CA94086-5206 (US).

(72) Inventors: **HENDERSON, Alex, E.**; 40 Denise Drive, Hillsborough, CA 94010 (US). **CROFT, Walter, E.**; 2311 Ticonderoga Drive, San Mateo, CA 94402 (US).

(74) Agents: **ARRIOLA-KERN, Trinidad, M** et al.; Fenwick & West LLP, Two Palo Alto Square, Palo Alto, CA 94306 (US).

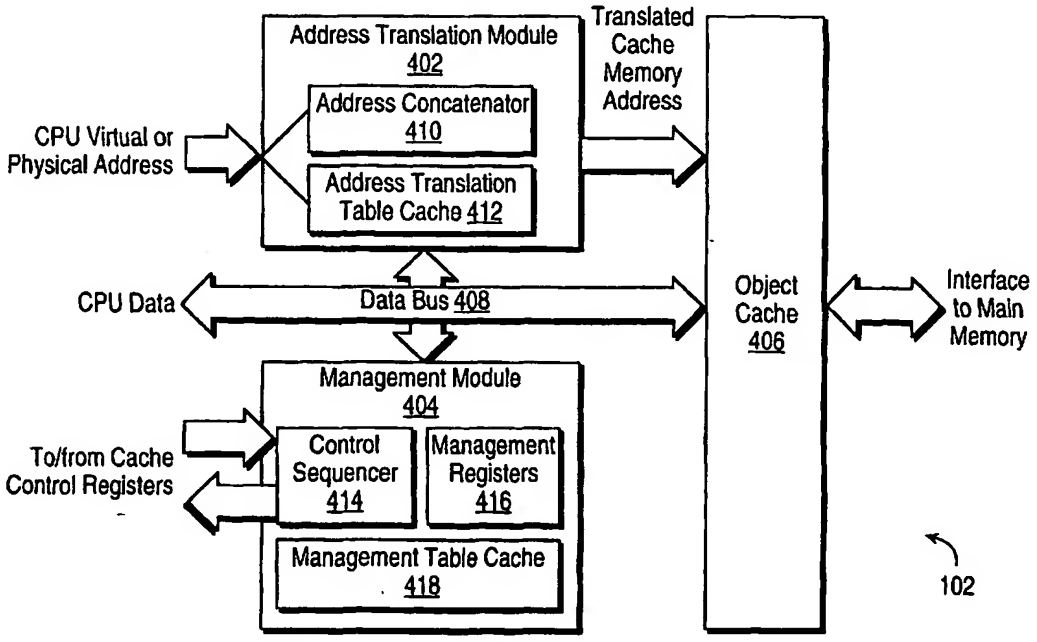
(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: DYNAMIC MEMORY CACHING



(57) Abstract: A system for mapping a sparsely populated virtual space of variable sized memory objects to a more densely populated physical address space of fixed size memory elements for use by a host processor comprises an object cache for caching frequently accessed memory elements and an object manager for managing the memory objects used by the host processor. The object manager may further comprise an address translation table for translating virtual space addresses for memory objects received from the host processor to physical space addresses for memory elements, and a management table for storing data associated with the memory objects used by the host processor.

WO 01/018653 A1

(48) **Date of publication of this corrected version:**
12 September 2002

(15) **Information about Correction:**
see PCT Gazette No. 37/2002 of 12 September 2002, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

DYNAMIC MEMORY CACHING

INVENTORS

Alex E. Henderson

Walter E. Croft

RELATED APPLICATION

The subject matter of the present application is related to and claims priority, under 35 U.S.C. §§ 120 and 119(e), from U.S. patent application serial no. 09/203,995, entitled "Dynamic Memory Manager with Improved Housekeeping" by Alex E. Henderson and Walter E. Croft, which application was filed on December 1, 1998 and is incorporated herein by reference in its entirety, and from U.S. provisional patent application serial no. 60/152,680, entitled "Dynamic Memory Caching" by Alex E. Henderson and Walter E. Croft, which application was filed on September 7, 1999 and is incorporated herein by reference in its entirety.

BACKGROUND

A. Technical Field

The present invention relates generally to computer memory allocation and management, and more particularly to efficiently managing the dynamic allocation, access, and release of memory used in a computational environment.

B. Background of the Invention

Historically, memory used in a computational environment, such as a computer, has been expensive and of questionable reliability. The general belief was that this memory should be utilized or "packed" as fully as possible. Methods for the efficient (here used in the sense of utilized) use of memory became standard, and have not been seriously questioned before this invention, though attempts have been made to reduce the impact on performance of such usage, and to make the operations more deterministic.

U.S. Patent Number 5,687, 368 ("the '368 patent") teaches the conventional view of the methods for efficient memory implementation. The '368 patent addresses a major shortcoming of the prior art, which is loss of computational performance due to the need for

memory management, also called housekeeping, to achieve efficient use of memory. The '368 patent teaches the use of a hardware implementation to alleviate the problem of loss of performance in the computational unit. However, the '368 patent does not teach reducing or eliminating housekeeping functions or mapping large, sparsely populated logical memory address space onto smaller, denser physical memory address space as in this invention. The '368 patent also does not teach making housekeeping functions more deterministic in the way or to the extent that the present invention does.

Traditional methods in the prior art, such as the '368 patent, copy data from memory location to memory location in order to compact and "garbage collect" the data. Garbage collection is a term used to describe the processes in a computer which recover previously used memory space when it is not longer in use. Garbage collection also consists of re-organizing memory to reduce the unused spaces created within the stored information when unused memory space is recovered, a condition known as fragmentation. The prior art inherently reduces the performance of the computational unit, due to the need to perform these operations and the time consumed thereby. Further, these operations are inherently not substantially deterministic, since the iterative steps required have no easily determinable limit in the number of iterations.

Basic assumptions in the prior art have been that memory should be optimized with respect to the utilization of the memory address space, rather than of the actual memory itself. Reliability was also considered to be a factor in utilizing available memory space as efficiently as possible. As a consequence, the atomic memory management data size was set in small blocks; usually 1024 bytes. Memory management systems (MMS) of the prior art then searched for memory not in use, often down to the individual block, so that memory space could be freed as expeditiously and to as small a unit size as possible.

The small size of the atomic memory unit often causes small pieces of memory, which are being used, to be interspersed with unused, or "garbage" locations, a process known as "fragmentation" of memory. Since this could result in significant problems in accessing streams of data due to the necessity to access small locations which are not contiguous, a technique known as "compaction" or "defragmentation" has been employed. This causes special commands and routines to be required and frequently used. In the UNIX operating system environment, when programming in ANSI C, for example. Function calls that directly or indirectly invoke these representative routines by allocating and releasing dynamic memory are known as "malloc()", "calloc()", "realloc()", and "free()". Again, these

functions and the directly or indirectly invoked representative routines require a substantially indefinite number of iterations, and are substantially not deterministic.

Additionally, to aid the functions above and to better utilize available memory, various concepts such as "relocatable memory" were developed and implemented, thereby allowing for more efficient routines for memory management functions such as compaction and defragmentation. Memory management functions, using relocatable memory, work by copying memory atomic units (objects) from one location in memory to another, to allow garbage fragments between valid objects to be combined into larger free memory areas. However, while improving the flexibility of the allocation process, relocatable memory also requires indefinite numbers of iterations, and further makes the time required for housekeeping functions substantially not deterministic. Accordingly, it is desirable to provide a system and method for a dynamic memory manager to overcome these and other limitations in the prior art.

Additionally, prior art memory management systems require extensive memory resources. None of the memory management systems in the prior art employ a caching technique. Caching is a process that stores frequently accessed data and programs in high speed memory local (or internal) to a computer processing unit for improved access time resulting in enhanced system performance. Caching relies on "locality of reference," the statistical probability that if a computer is accessing one area of memory that future accesses will be to nearby addresses. A cache gains much of its performance advantage from the statistical probability that if a computer is accessing one part of an object that future accesses will be to other parts of the same object. Cache memories are classified by the type of association used to access the data (e.g. direct mapped, set associative, or fully associative), the replacement algorithm (e.g. Least Recently Used ("LRU") or Least Frequently Used ("LFU"), and the write algorithm (e.g. write back or write through). Cache memories are typically much smaller than the main system memory. The size of a cache memory, type of association, and access statistics of the program(s) executing determine the probability that a piece of data is in the cache when an access to that data occurs. This "hit rate" is a key determinant of system performance.

Accordingly it is desirable to provide a system and method for dynamic memory management technology in conjunction with caching techniques to reduce on chip memory requirements for dynamic memory management.

SUMMARY OF THE INVENTION

The present invention overcomes the deficiencies and limitations of the prior art with a novel system and method for dynamic memory management technology. A system for dynamic memory management maps a sparsely populated virtual address space of memory objects to a more densely populated physical address space of fixed size memory elements for use by a host processor. In one aspect, the system comprises an object cache for caching frequently accessed memory elements and an object manager for managing the memory objects used by the host processor. The object manager may further comprise an address translation table for translating virtual space addresses for a memory object received from the host processor to a physical space address for a memory element, and a management table for storing data associated with the memory objects and memory elements. In one embodiment, the address translation table and the management table are stored in the physical system memory. In another embodiment, the present invention further comprises an address translation table cache for caching the most recently or most frequently used address translation table entries. In yet another embodiment, the present invention further comprises a management table cache for caching the most recently or most frequently used management table entries.

In another aspect, a method for mapping a memory object used by a host processor to a memory element stored in physical memory comprises the steps of receiving a virtual space address for a memory object used by a host processor, determining a physical space address for the memory element or elements in the memory object, and retrieving the memory element from the physical system memory. In one embodiment, the present invention first checks the object cache to determine whether the memory element has been cached. If the memory element is in the object cache, it is an object cache "hit". If the memory element is not stored in the object cache, it is an object cache "miss", and the memory element is retrieved from physical system memory and stored in the cache according to the cache replacement logic.

These and other features and advantages of the present invention may be better understood by considering the following detailed description of preferred embodiments of the invention. In the course of this description, reference will be frequently made to the attached drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a high level block diagrams of one embodiment of a system in accordance with the present invention.

Figures 2A-2C are high level block diagrams of other embodiments of systems in accordance with the present invention.

Figure 3A is a dynamic memory mapping diagram in accordance with one embodiment of the present invention.

Figure 3B is another embodiment of the present invention comprising caching associative memories.

Figure 4 is a block diagram of one embodiment of a Dynamic Memory Cache in accordance with the present invention.

Figure 5 is a block diagram illustrating additional details of the management module 404.

Figure 6 is a flow chart of one embodiment of the main loop process for the control sequencer 414.

Figure 7 is a flow chart of one embodiment of the initialize process for the control sequencer 414.

Figure 8 is a flow chart of one embodiment of the allocate process for the control sequencer 414.

Figure 9 is a flow chart of one embodiment for a release process for the control sequencer 414.

Figure 10 is a flow chart of one embodiment of the diagnostic process of the control sequencer 414.

Figure 11 is a block diagram of one embodiment of an aging process for a Least Recently Used (LRU) replacement algorithm.

Figure 12 is a block diagram of an LRU replacement algorithm implemented using a distributed implementation of an aging circuit.

Figure 13 is a block diagram of a single distributed oldest circuit.

Figure 14 is a functional block diagram of one embodiment of an address translation module 402.

Figure 15 is a block diagram of the address concatenator 410.

Figure 16 is a flow chart of one embodiment for allocating and releasing a memory object in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to Figure 1, there is shown a block diagram of a system in accordance with the present invention. The present invention comprises a Dynamic Memory Cache ("DMC") 102 coupled to a host processor 104 and to other memory 106. In a preferred embodiment, the host processor 104 has a level 1 cache. The other memory 106 may comprise a RAM, ROM, Flash or other memory or may comprise other devices such as a disk, video, network, etc... The present invention provides a dynamically allocated memory object (not shown) for use by the host processor 104. The memory object comprises a plurality of memory elements or locations in other memory 106. The present invention maps the memory object used by the host processor 104 to a plurality of memory elements in the other memory 106. The memory elements are memory locations of fixed size in the other memory 106. For example, memory elements may be 16 bytes or they may be 64 bytes. The DMC 102 manages the memory objects used by the host processor 104 and performs the address translation functions between the host processor 104 and the other memory 106. Memory objects and memory object mappings are described in detail in copending application serial no. 09/203,995 entitled "Dynamic Memory Manager with Improved Housekeeping" by Walter E. Croft and Alex E. Henderson, which application was filed on December 1, 1998, and which application is incorporated herein by reference in its entirety. Thus, the present invention advantageously allocates memory objects to the host processor 104 from a large sparsely populated virtual memory space and maps the allocated memory objects to a smaller densely populated physical memory space. This mapping provides the basis for the removal of dynamic memory housekeeping functions such as "garbage collection", de-fragementation, and compaction.

Referring now to Figure 2, there is shown a high level block diagram of another embodiment of a system in accordance with the present invention. The present invention comprises a DMC 102 coupled to CPU or host processor 204 and to a bus interface 206 to a separate memory location. The DMC 102 further comprises an object manager 208 for allocation, de-allocation, and control of caching of the memory elements, and an object cache 210 for the storage of cached memory elements. Figure 2A also shows a conventional data cache 212, conventional data Translation Lookaside Buffer (TLB), a conventional instructional cache 214, and instruction Translation Lookaside Buffer (TLB) to illustrate the high level similarities between the operation of the DMC with respect to the CPU 204 and the

bus interface 206. Figures 2B and 2C illustrate various useful combinations of conventional TLB and caching with object management and object caching. These are analogous to conventional combined or "unified" instruction and data TLB and caches and offer the benefits of shared TLB tables and caches while maintaining the benefits of object management and object caching.

Referring now to Figure 3A, there is shown a dynamic memory mapping diagram in accordance with one embodiment of the present invention. The present invention comprises a host processor virtual address space 304 for storing the memory objects 308A, 308B, and 308C, that are used by the CPU or host processor. Each memory object is mapped to one or more memory elements located in the physical system memory 306. For example, memory object 308A is mapped to three memory elements and memory object 308B is mapped to one memory element. The virtual space address of the memory object 308 used by the host processor is inputted to the DMC 102 for translation by the address translation module 310. The address translation module 310 translates virtual space addresses for memory objects 308 to physical space addresses for memory elements. In a preferred embodiment, the memory element is stored in the object cache 210 and can be accessed using the physical space address for the memory element. If the host processor accesses a memory element not found in the object cache 210, a miss will occur and the object manager 208 will replace entries in the management table, address translation table, and object cache to provide access to the desired object.

The DMC 102 maintains large software management and address translation tables in physical system memory 306. These large tables allow the management of very large numbers of objects. In one embodiment, physical system memory 306 maintains four data structures: a memory element table 312, a management table 314, an address translation table 316, and a process table 318. The memory element table 312 is a pool of small fixed sized memory areas ("memory elements") used to store data. These memory areas may or may not be sequentially located in memory. In one embodiment, these memory areas may be partitioned into multiple separate pools of memory elements allocated on a per process basis.

Management table 314 refers to a table or group of tables that store information about the size and address translation table entries of each allocated memory object. The management table 314 may be organized as an AVL tree, a hash table, a binary tree, a sorted table, or any other organizational structure that allows for rapid search and insertion and

deletion of entries. In another embodiment, the most frequently used or most recently used management table entries are stored in a management table cache.

Address translation table 316 refers to a table or group of tables that store the virtual to physical address translation information for each memory element. In one embodiment, a single memory object will typically use several address translation table entries. In a preferred embodiment, the address translation table 316 may be organized as an AVL tree, a hash table, a binary tree, a sorted table, or any other organizational structure that allows for rapid search and insertion and deletion of entries. In another embodiment, the most frequently used or most recently used address translation table entries are stored in an address translation table cache.

The process table 318 refers to a table sorted by process, program, or thread ID that is used to locate the management table entries for memory objects associated with a particular process, program, or thread. In a preferred embodiment, this table is organized as an AVL tree to allow for rapid search and insertion and deletion of entries.

Referring now to Figure 3B, there is shown another embodiment of the present invention. The embodiment in Figure 3B uses caching associative memories to implement the management table and the address translation table. Caching associative memories are described in more detail in copending U.S. patent application serial number _____, entitled "Caching Associative Memories" by Alex E. Henderson and Walter E. Croft, which application was filed on August 10, 2000 and which application is incorporated herein by reference in its entirety. More specifically, in this embodiment, the management table 326 is stored in a main associative memory and the address translation table 324 is stored in a main associative memory. The most frequently used or most recently used management table entries are stored in a management table associative memory cache 322. Similarly, the most frequently used or most recently used address translation table entries are stored in an address translation table associative memory cache 320. Associative memory caches have replacement logic to manage the replacement of cached data as explained in U.S. patent application serial no. _____.

In one embodiment, the present invention may be used in an operating system application. In a typical operating system application, there will be a large pool of object memory. The management table 314, address translation table 316, and process table 318 can be dynamically allocated supervisor or system privilege level objects. At system reset, the memory element table 312 would be initialized to contain three objects: management

table 314, address translation table 316, and process table 318. The process table 318 will initially contain only one entry, the supervisor or system process entry. It may point to a management table that contains three entries, the process table, management table and address translation table entries. The address translation table may contain entries sufficient to define the physical address of these objects. A user process can request the allocation of a variable sized memory object from the operating system. The operating system, supervisor, or system process then dynamically allocates space for a new management table entry (an object belonging to the system process) and as address translation table entries (also belonging to the system process) as required to describe the requested object. The user process can then access the new memory object. Deallocation is the reverse process of deallocating the system objects used for the address translation and management table entries.

Referring now to Figure 4, there is shown a block diagram of one embodiment of a DMC 102 in accordance with the present invention. The DMC 102 comprises an address translation module 402, a management module 404, and an object cache 406. The address translation module 402 and management module 404 communicate directly with the CPU or host processor, and are coupled to the object cache 406 via data bus 408.

The management module 404 manages the object cache 406 and address translation module 402 for the DMC. The management module 404 preferably comprises a control sequencer 414, management registers 416, and a management table cache 418. Control sequencer 414 scans the CPU registers (not shown) for host processor commands, executes valid commands, and loads results for the host processor 104. Management table cache 418 contains an entry for each memory object active in the DMC 102.

The address translation module 402 translates the CPU virtual space address for a memory object to a physical memory space address for a memory element. The address translation module 402 comprises an address concatenator 410 and an address translation table cache 412. The address translation table cache 412 performs the content addressable memory ("CAM") lookup of object base address and object block index bits of the host processor virtual space address for the memory object, as described in more detail with reference to Figure 14. If a valid cache entry exists for the physical address of the memory element, the address translation table cache 412 provides a cache address and physical memory address. The address translation table cache 412 contains memory element information comprising an object base address, which is known to the management table cache 418, an object block index, which is a secondary portion of the base address, a link to

the next object base address/block index pair, a link back to the management table 418 entry for this object, an address of segment in cache, and an address of segment in system memory. The address concatenator 410 receives the address of the segment in cache from the address translation table cache 412. The address concatenator 410 also receives pass through low order bits of the host process address. The address concatenator 410 then concatenates the cache address and pass through low order bits and generates the cache memory address for the object cache 406.

The object cache 406 provides a fast local memory used to store frequently accessed memory element data. The cache replacement logic for object cache 406 selects the cache line or lines to be replaced in case of management table cache 418 or address translation table cache 412 misses. In a preferred embodiment, the object cache 406 uses a Least Recently Used ("LRU") replacement algorithm. The object cache 406 may include a write buffer to implement a delayed write of altered object data to other memory 106. The write may be a single word for write through caching or a complete object cache line buffer for write back caching. Write back and write through may be a selectable mode. In another embodiment, optional object cache coherency logic may be used for monitoring system bus writes by other devices to shared objects. The coherency logic may implement any of the classical bus snooping and cache coherency schemes.

Referring now to Figure 5, there is shown a block diagram illustrating additional details of the management module 404. Individual entries in the Management table cache 418 comprise an Object Start Address 502, Object Size 504, Process ID 506, Age and Dirty Flag 508, and Object Number 510. Management table cache 418 may also contain optional user and system data. In a preferred embodiment, a Least Recently Used ("LRU") algorithm is used to determine which management table cache 418 entry to replace. When an object is accessed that does not have a management table entry, the event is considered a "management table miss".

Figure 5 also shows an example of three dynamically allocated memory objects of varying size added after DMC initialization. The object start address 502 and the object size 504 of the three memory objects define the location and extent of the memory objects in the virtual address space of the process specified by the process ID 506. Object number field 510 provides the index to the management table 314. In one embodiment, Age and Dirty Flag 508 and object number 510 are used to implement a LRU replacement algorithm. Preferably, all ages 508 are set to zero and dirty flags 508 are cleared by a system reset. When a new

entry is added to the management table cache 418 the oldest entry (e.g. Age=0) is replaced. If more than one entry has an age of 0, the entry with the largest object number 510 is replaced. If the dirty flag 508 is set (for example, as a result of a re-alloc operation or a write to the object) the replaced entry is written back to other memory 106.

Management registers 416 provide working data for the DMC. These registers contain information about the address translation module 402 and the management module 404. The management registers 416 contain results of host processor commands that are returned to the host via the user registers. Management registers 416 comprise a set of permanent registers 512 and temporary registers 514. The permanent registers 512 contain information such as the maximum size of a memory object, the number of free entries in the management table cache 418, a pointer to the next free entry in the management table cache 418, the number of free entries in the address translation table cache 412, and a pointer to the next free entry in the address translation table cache 412. Preferably, the permanent registers 512 are initialized at power on and reset. Temporary registers 514 contain information such as the memory size requested, the calculated number of address translation table cache entries, and pointers, counters, etc... .

The control sequencer 414 is the processing core of the DMC 102 and utilizes control and status signals to access each section of the DMC via the internal data bus 408. The control sequencer 414 comprises at least five different control sequences: 1) main loop, 2) initialize process, 3) allocate process, 4) release process, and 5) diagnostic process. The main loop process of the control sequencer 414 executes the initialize process on power up or reset, monitors the user device control register (not shown) for commands, dispatches the command for execution, and makes available the results of the command to the host processor. The initialize process sets the DMC and associated private memory to a known state. The allocate process verifies that the dynamic memory allocation is valid and claims resources, adds memory objects, and updates status. The release process verifies that the dynamic memory release is valid and frees resources, removes memory objects, and updates status. The diagnostic process reads or writes a specified DMC data element.

Referring now to Figure 6, there is shown a flow chart of one embodiment of the main loop process for the control sequencer 414. This process is started by a system reset. After the system reset, the initialize process initializes the DMC. After initialization is complete, the control sequencer pools the device control register for a command. When a command is detected, the busy indication is set in the device status register 606. The command is decoded

to determine which sub process should run. If no valid command is found, the command error bit in the device status register is set 626, otherwise the command results bits in the device status register are set 624 on sub process completion. The busy indication in the device status register is then cleared 628 and the contents of the user registers are available 230 to the CPU.

Referring now to Figure 7, there is shown a flow chart of one embodiment of the initialize process for the control sequencer 414. The process starts at 702 and builds a free list of address translation table cache entries 704. The process then builds a free list of management table cache entries 706. Next, the process initializes the management registers 708 and ends at 710.

Referring now to Figure 8, there is shown a flow chart of one embodiment of the allocate process for the control sequencer 414. The process starts at 802 and determines 804 whether a management table cache entry is free. If an entry is not free, the device status register is set to indicate allocate an error 806 and the process ends 818. If an entry is free, the process then determines 808 whether an address translation table cache entry is free. If an entry is not free, the device status register is set to indicate an allocate error 806 and the process ends 818. If an entry is free, the process gets an entry from the management table cache free list and adds the management table cache entry 810. The process then gets entries from the address translation table cache free list and adds and links address translation table cache entries 812. The process then updates 814 the management registers. Finally, results of the allocate are stored in the device status register and the allocated object is available for use 816.

Referring now to Figure 9, there is shown a flow chart of one embodiment for a release process for the control sequencer 414. The process starts at 902 and determines 904 whether the management table cache entry has been found. If the answer is no, the device status register indicates a "release error" 906 and ends at 918. If the management table cache entry is found, the process then determines 908 whether the address translation table cache entries can be found. If the answer is no, the device status register indicates a "release error" 906 and the process ends 918. If the answer is yes, the process deletes the management table cache entry and returns the entry to the management table free list 910. The process then deletes the address translation table entries and returns the entries to the address translation table free list 912. Afterwards, the process updates 914 the management registers. The

device status register then indicates 916 the release results and indicates that the released object is not accessible.

Referring now to Figure 10, there is shown a flow chart of one embodiment of the diagnostic process of the control sequencer 414. The diagnostic process provides software access to the internal data structures of the DMC for software diagnostics. Sub commands are provided to read and write the Address Translation Table cache 412, Management Table cache 418, and Management Registers 416. These commands are decoded by decisions 1002. The parameters for these commands are validated by the decisions 1004. If either a bad sub command or invalid parameter is detected the diagnostic error indication in the device status register is set. If the sub command and parameters are valid, the read or write function 1006 is executed and the read or write result set in the device status register is set 1010.

Referring now to Figure 11, there is shown a block diagram of one embodiment of a LRU cache replacement logic. The entry match logic compares 1102 the process ID and virtual address from the CPU with the values stored in the management table cache process ID, object start address 502 and object size 504. If there is a match a management table cache hit has occurred and the ages of the management table cache entries must be updated. The age process 1104 works as follows: The age of the management table cache entry for which the hit occurred is driven 1106 on the current age bus. The age of any entry with an age greater than the current age is decremented. The age of the management table cache entry for which the hit occurred is set to the number of management table cache entries minus one. The other age entries are unchanged. If a miss occurs (no hit occurred) the management table in system memory 314 is searched. If a match is found, the oldest entry in the management table cache 418 is replaced.

Referring now to Figure 12, there is shown an implementation of the age update process where the comparison of each management table cache entry's age is compared to the current age by duplicated compare circuits 1202. These circuits determine which entries ages should be decremented, which should stay the same (no operation or no-op) and which one should be loaded with the total number of management table entries minus one.

Referring now to Figure 13, there is shown a block diagram of implementation of a distributed compare circuit. The row with a hit drives the current age bus. All rows compute the greater than and equal to signals. These signals control which ages are decremented or loaded with the total number of management table entries minus one.

Referring now to Figure 14, there is shown a functional block diagram of one embodiment of an address translation module 402 block diagram. As discussed above with reference to Figure 4, the address translation module 402 comprises an address concatenator 410 and an address translation table cache 412. As shown in Figure 14, the address translation table cache 412 comprises a content addressable memory ("CAM") 1402 for enabling fast searches and associated data 1404 for providing entry specific information. One skilled in the art will realize that a CAM and associated data are not the only suitable devices for an address translation table but that any type of associative memory, which allows searches based on content as opposed to address location, may be used for the address translation table cache 412, and that the description here of a CAM and associated data are for illustrative purposes only.

The operation of the address translation module 402 is as follows. The host processor addresses 1406 are placed on the host processor address bus 1406 and are detected and used as input to the address translation module 402. In one embodiment, the DMC address range is a 32 bit address range with the high-order 26 bits being utilized for translation and the low-order 6 bits being passed on directly. The passed on 6 bits define a maximum segment offset size of 64 bytes. Match Data 1408, which in this embodiment is the high order 26 bits, is extracted from the host processor address 1406 and subdivided into two sections for searching the CAM: a base address 1410 and a block index 1412. If a search on the CAM results in a "miss" (i.e. match data is not located in CAM), then an address translation table cache entry must be loaded. Additionally, a new management table cache entry may also be required. When a search on the CAM 1402 using the Match Data 1408 results in a match, a corresponding match signal 1414 for the CAM entry is asserted for specifying a particular entry in the associated data 1404. Individual entries in the associated data 1404 that comprise a single memory element are linked together by a link field 1416. Unused entries are part of the address translation table cache free list. Active entries in the associated data 1404 also have a management table link 1418 for providing a link to the management table cache 418. Unused links are nullified. If a link field 1416 is NULL, signaling that this is the final segment of this memory element, the management table link is used to determine memory object size 504 in bytes. The valid byte length of the ending segment can be calculated by the modules of the object size 504 by the memory element size. The remainder of bytes in the last memory element will range from 1 to the memory element size. In one embodiment, only part of the addresses in this ending segment may be valid. If part of the addresses are

invalid, an invalid address bus error is generated to alert the host processor. Translated cache addresses are stored in the mapped address field 1420. Translated cache addresses are determined during initialization and are treated as read-only data elements during operation of the present invention. The cache address 1420 associated with the match data 1408 search are then passed to the address concatenator 410. Thus, validated host processor addresses 1406 enable the mapped address to be concatenated with the pass through low-order 6 bits of the host processor address 1406 to form the translated cache memory address, thereby providing access to the memory object in the cache memory.

Referring now to Figure 15, there is shown a block diagram of the address concatenator 410. As discussed above with reference to Figure 14, the host processor address 1406 is placed on the system bus and used as an input to the address translation module 402. In general, bits N+1-M are used as the address bits for translation and are used to search the CAM 1402. The value N determines the size of an object cache line. For the example below, N is equal to 5. The value M is the width of the CPU address bus. For the example below, M is 32. In one embodiment, the high-order 26 bits are utilized for translation. Bits 0-N are passed on directly to the address concatenator 410. In one embodiment, the low-order 6 bits are the passed on bits. One skilled in the art will realize that the subdivision of the host processor address 1406 into bits used for translation and pass through bits is not limited to the examples provided here but may be subdivided as necessary or desired for utilization of the invention. For example, the low order 16 bits may be used for translation and the high order 16 bits may be used for passing through to the concatenator. The translated bits (Bits N+1-L) are then retrieved from the address translation table 412 as described with reference to figure 14 and concatenated with the pass through bits (Bits 0-N). The newly concatenated translated bits (Bits N+1-L) with the pass through bits (Bits 0-N) are then sent to the managed address space 1506.

Referring again to Figure 14 and address translation table cache 412, there is shown an example of the linked segments that support three dynamically allocated memory object in the management table cache 418. From the example, management table cache entry 1 is added at the bottom of the address translation table cache 412. The base address field 1410 for this entry starts at 80000000 hexadecimal or 2^{31}, and the block index field 1412 starts at 0 and increases by 100 hex (256 bytes). Following the management table link 1418 of 1, the management table memory allocate size field shows a memory object of 514 bytes. 514 bytes fits in three 256 byte segments that are connected by the link field 1416 with values of

1, 2, and NULL to end the list of segements. The translated cache memory address 0, 100, and 200 hex are the cache memory addresses 1420 for the 514 byte memory object. In one embodiment, the translated cache addresses are on 256 byte boundaries at offsets 0, 256, and 512 bytes respectively. Management table cache entry 2 is added above management table cache entry 1 in this example. For entry 2, the base address starts at 80010000 hex which is 65,536 bytes above the start address for management table cache entry 1. Thus, in this example, this sets the maximum individual memory object size of 65,536 bytes built from 256 address translation table entries.

Referring now to Figure 16, there is shown a flow chart of one embodiment for allocating and releasing a memory cache object in accordance with the present invention. A memory cache object is allocated by first creating or removing 1602 a management table cache entry for the object in the management table cache 418 for the currently executing process, program, or thread. Then, the address translation cache entries for the memory element in the address translation table are created or removed 1604 for the currently executing process, program, or thread. Finally, the new address translation table cache entries are pointed 1606 at the memory allocated from the memory element pool. Alternatively, the allocated memory may be returned to the memory element pool.

From the above description, it will be apparent that the invention disclosed herein provides a novel and advantageous method and system for dynamically allocating cached memory objects to a host processor. The foregoing discussion discloses and describes merely exemplary methods and embodiments of the present invention. As will be understood by those familiar with the art, the invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. Accordingly, the disclosure of the present invention is intended to be illustrative, but not limiting, of the scope of the invention, which is set forth in the following claims.

CLAIMS

We claim:

1. A system for mapping a sparsely populated logical address space of memory objects to a more densely populated physical address space of fixed size memory elements for use by a host processor, the system comprising:

an object cache for caching frequently accessed memory elements; and

an object manager for managing the memory elements stored in the object cache.

2. The system of claim 1 wherein the object manager further comprises:

an address translation table for translating a memory object address received from the host processor to an address of a fixed sized memory element in the object cache or physical memory.

3. The system of claim 1 wherein the object manager further comprises:

a management table for storing data associated with the memory objects used by the host processor.

4. The system of claim 3 wherein the management table is stored in physical memory.

5. The system of claim 3 wherein the management table is organized as an AVL tree.

6. The system of claim 3 wherein the management table is organized as a hash table.

7 The system of claim 3 wherein the management table is organized as a binary tree.

8. The system of claim 3 wherein the management table is organized as a sorted list.

9. The system of claim 3 further comprising a management table cache for storing the most recently or most frequently used management table entries.

10. The system of claim 9 wherein the management table cache is an associative memory cache.

11. The system of claim 2 wherein the address translation table is stored in physical memory.

12. The system of claim 2 wherein the address translation table is organized as an AVL tree.

13. The system of claim 2 wherein the address translation table is organized as a hash table.

14. The system of claim 2 wherein the address translation table is organized as a binary tree.

15. The system of claim 2 wherein the address translation table is organized as a sorted list.

16. The system of claim 2 further comprising an address translation table cache for storing the most recently or most frequently used address translation table entries.

17. The system of claim 16 wherein the address translation table cache is an associative memory cache.

18. A system for mapping a sparsely populated logical address space of memory objects to a more densely populated physical address space of fixed size memory elements, the system comprising:

an object cache, coupled to the physical address space, for caching a plurality of memory elements; and

an address translation module, coupled to the object cache and to the logical address space, for receiving a memory object address from a host processor and for translating the memory object address into a memory element address.

19. The system of claim 18 further comprising:

a management module, coupled to the address translation module, for storing data associated with the memory objects.

20. The system of claim 18 wherein the address translation module further comprises:

an address translation table and an address translation table cache for receiving a virtual space address for a memory object and for translating the virtual space address to a physical memory address for the memory element; and

an address concatenator coupled to receive pass through bits from the virtual space address and for concatenating the pass through bits with the physical memory address.

21. The system of claim 19 wherein the management module further comprises:

a management table and a management table cache for storing data associated with the memory object;

a control sequencer for scanning and executing host processor commands; and

a plurality of management registers for storing information associated with the address translation module and the management translation module.

22. The system of claim 20 wherein the address translation table cache comprises a CAM.

23. The system of claim 11 wherein the management table cache comprises a CAM.

24. A method for mapping a virtual space address for a memory object used by a host processor to a physical space address for a memory element stored in a memory management system, the memory management systems comprising an object cache, for caching frequently or recently accessed memory elements, and an object manager, for storing data associated with the memory object, the method comprising:

receiving a virtual address for a memory object used by a host processor;

determining a physical address for the memory element in the memory object; and

retrieving the memory element from the object cache.

25. The method of claim 24 wherein the address for the memory object further comprises a plurality of translated bits and a plurality of pass through bits, and further comprising the steps of:

determining a cache address for the translated bits; and

concatentating the cache address with the pass through bits.

26. The method of claim 24 wherein the object manager further comprises an address translation module for storing cache addresses for memory objects stored in the object cache.

27. The method of claim 25 wherein the address translation module comprises a CAM.

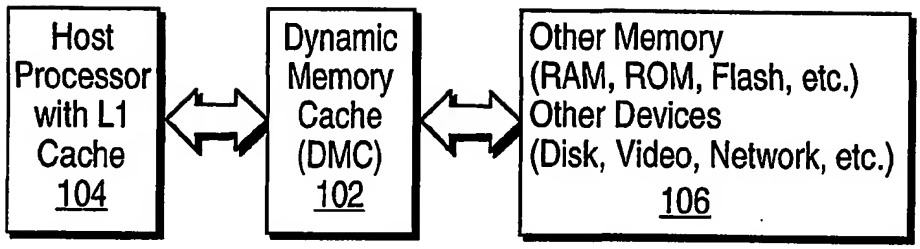

Host Processor with L1 Cache 104
Dynamic Memory Cache (DMC) 102
Other Memory (RAM, ROM, Flash, etc.)
Other Devices (Disk, Video, Network, etc.) 106

FIG. 1

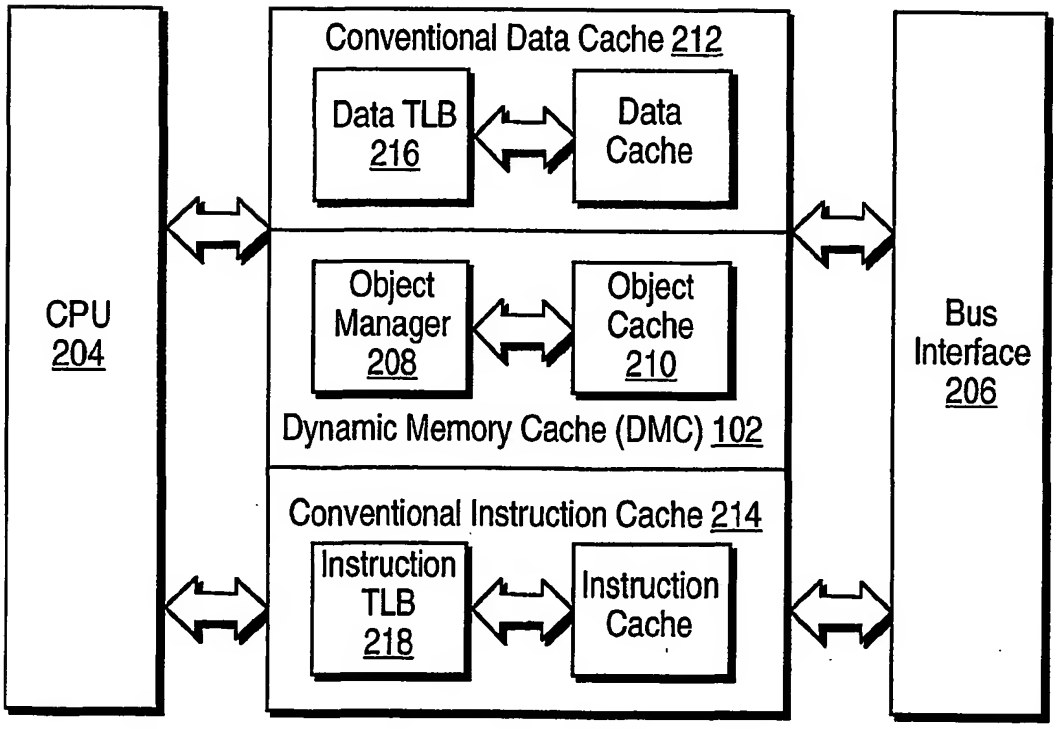

CPU 204
Conventional Data Cache 212
Data TLB 216
Data Cache
Object Manager 208
Object Cache 210
Dynamic Memory Cache (DMC) 102
Conventional Instruction Cache 214
Instruction TLB 218
Instruction Cache
Bus Interface 206

FIG. 2A

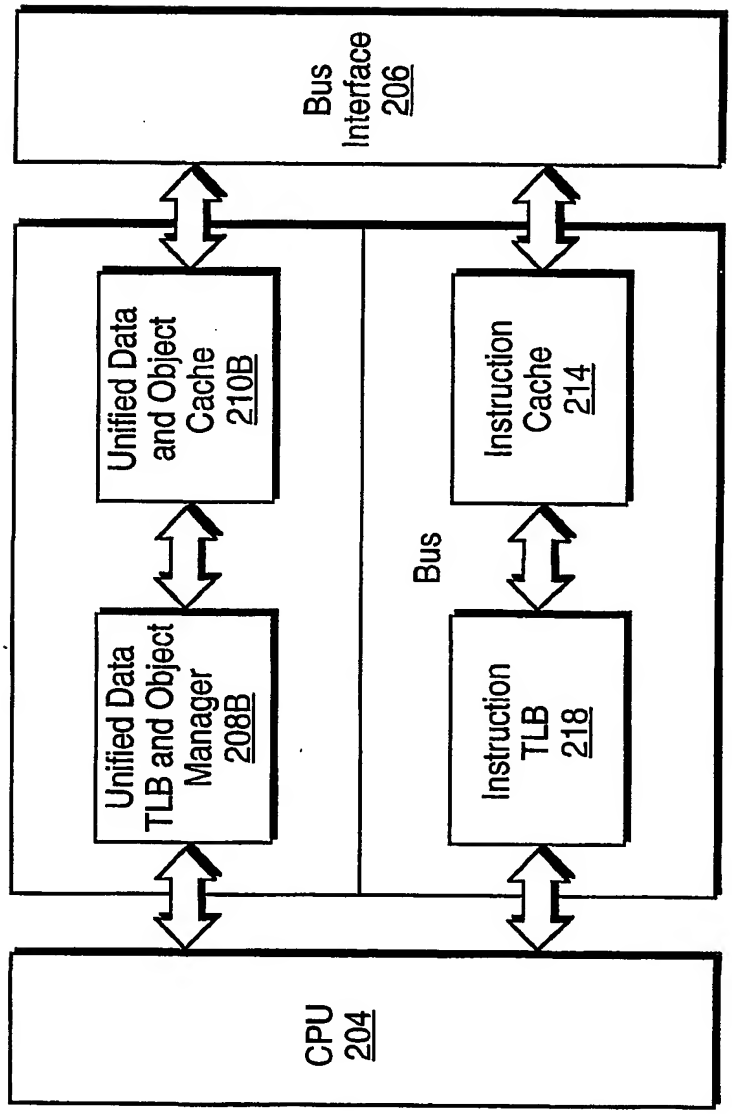

Bus Interface 206
Unified Data and Object Cache 210B
Unified Data TLB and Object Manager 208B
Instruction Cache 214
Instruction TLB 218
Bus
CPU 204

FIG. 2B

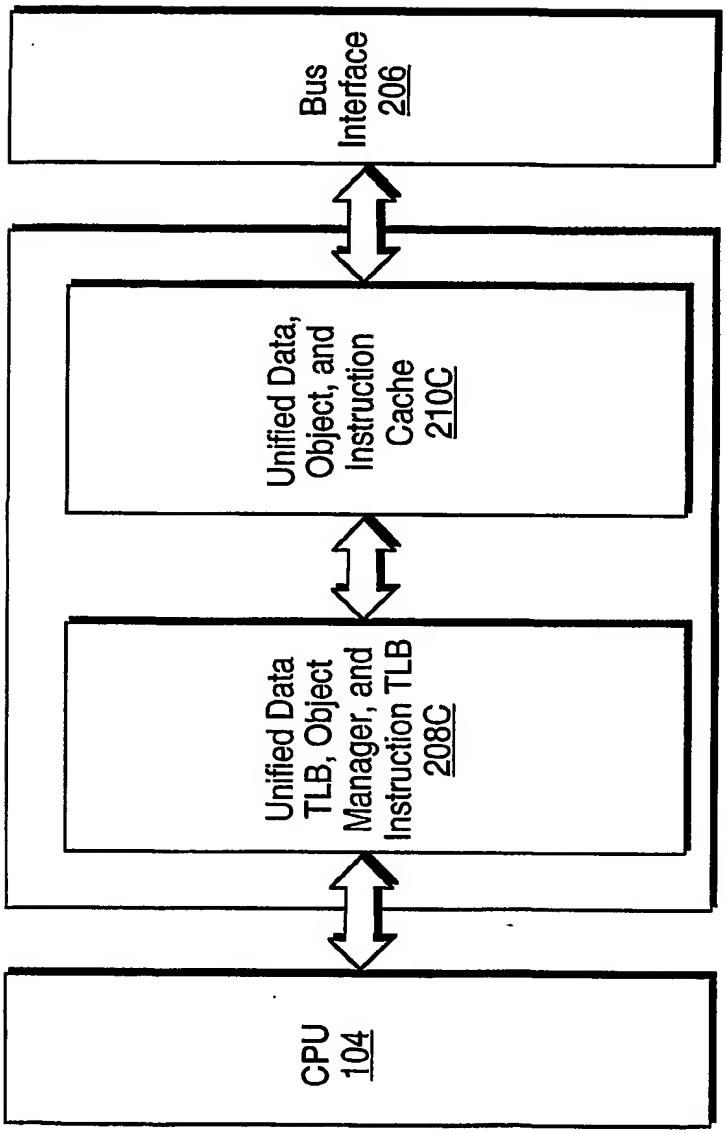

Bus Interface 206
Unified Data, Object, and Instruction Cache 210C
Unified Data TLB, Object Manager, and Instruction TLB 208C
CPU 104

FIG. 2C

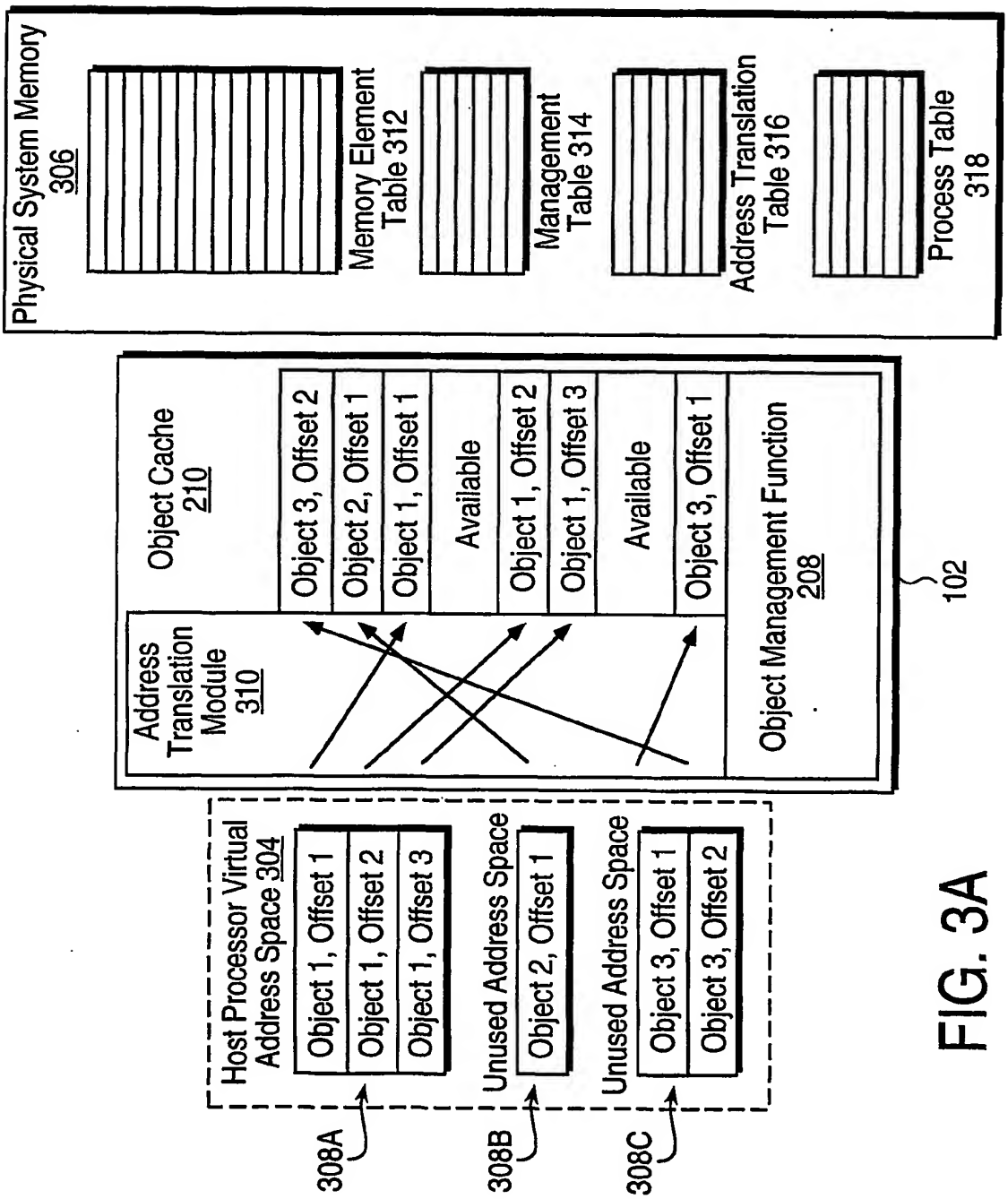

Physical System Memory 306
Memory Element Table 312
Management Table 314
Address Translation Table 316
Process Table 318
Object Cache 210
Object 3, Offset 2
Object 2, Offset 1
Object 1, Offset 1
Available
Object 1, Offset 2
Object 1, Offset 3
Available
Object 3, Offset 1
Address Translation Module 310
Object Management Function 208
102
Host Processor Virtual Address Space 304
Object 1, Offset 1
Object 1, Offset 2
Object 1, Offset 3
Unused Address Space
Object 2, Offset 1
Unused Address Space
Object 3, Offset 1
Object 3, Offset 2
308A
308B
308C

FIG. 3A

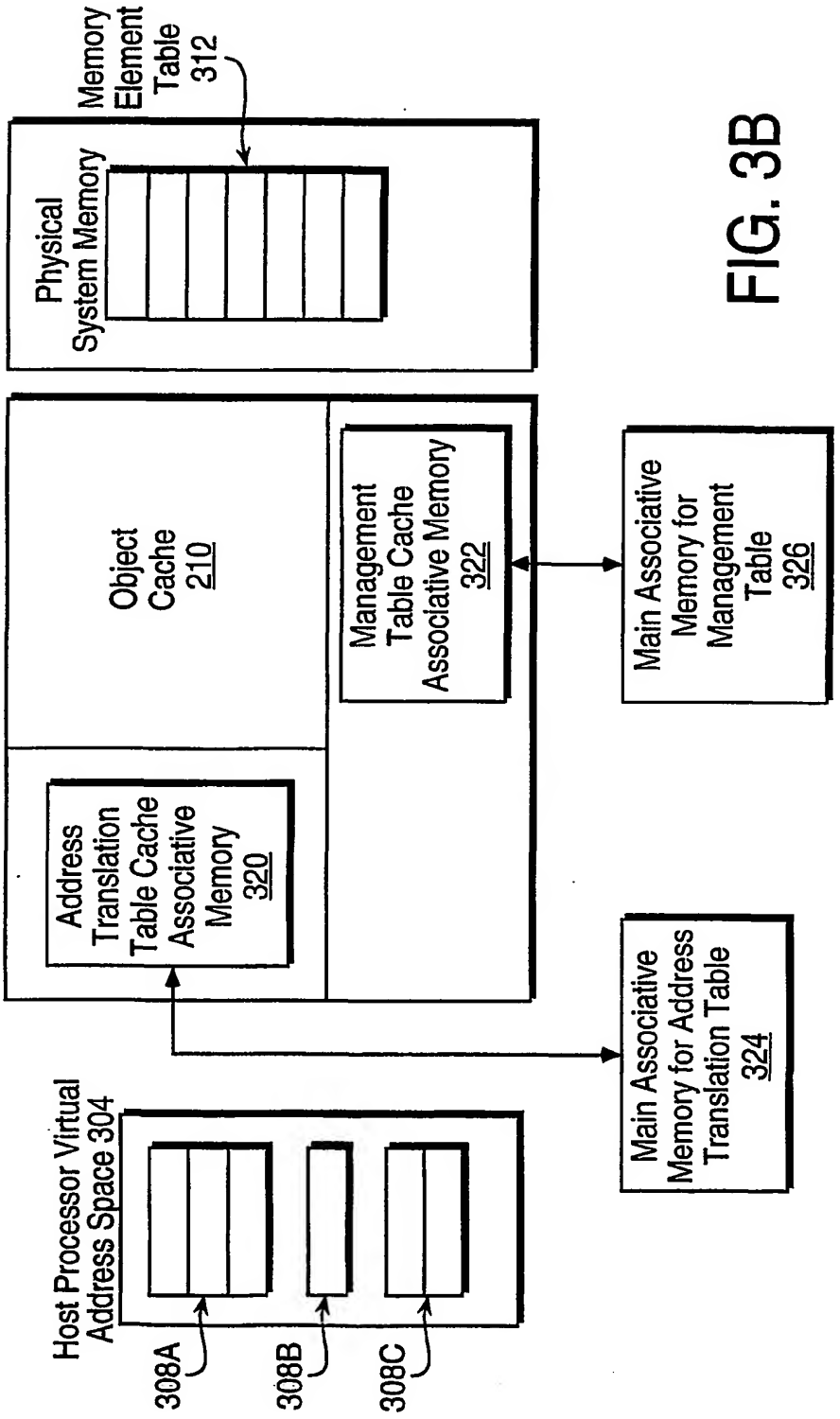

Memory Element Table 312
Physical System Memory
Object Cache 210
Management Table Cache Associative Memory 322
Main Associative Memory for Management Table 326
Address Translation Table Cache Associative Memory 320
Main Associative Memory for Address Translation Table 324
Host Processor Virtual Address Space 304
308A
308B
308C

FIG. 3B

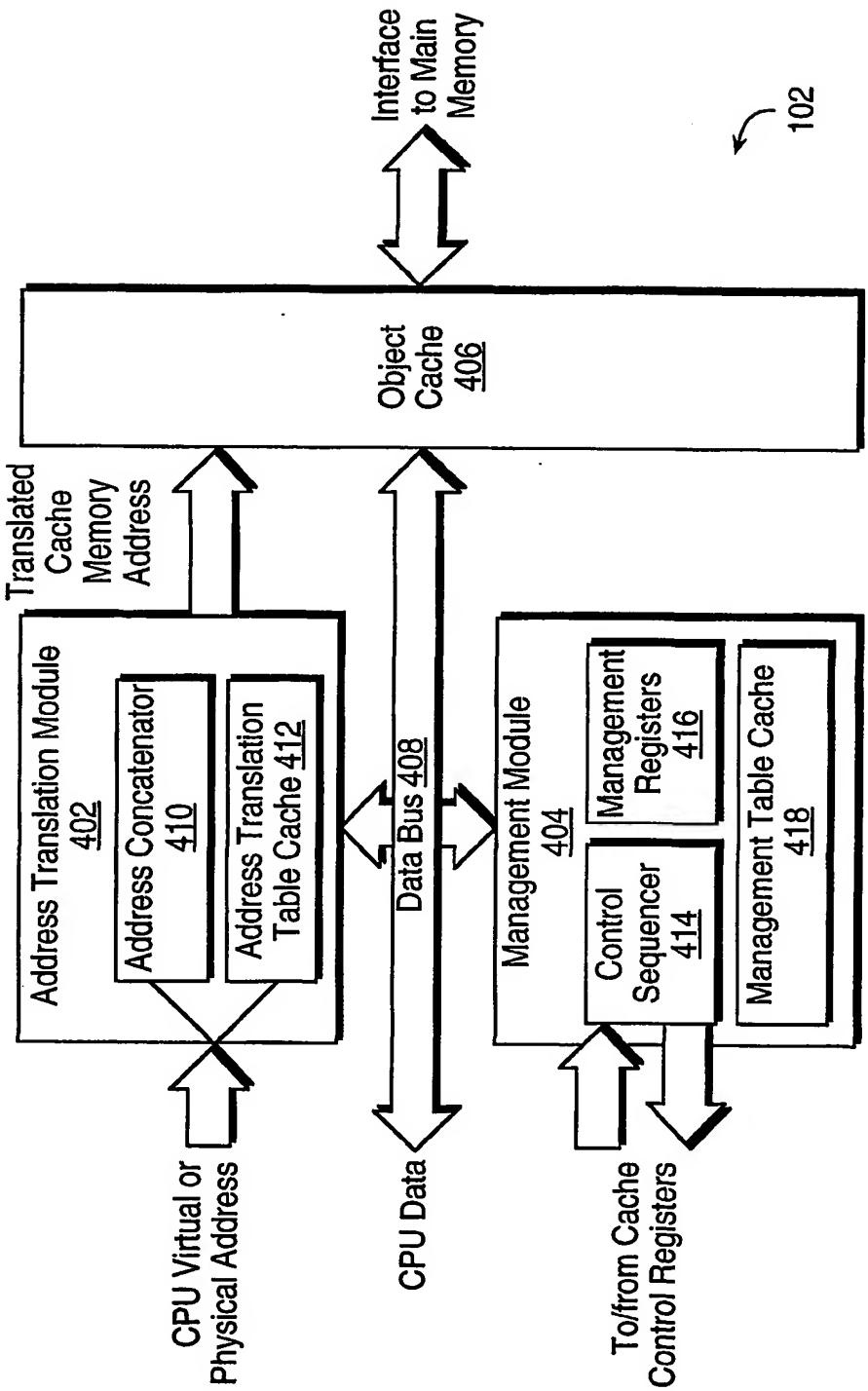

Interface to Main Memory
102
Object Cache 406
Translated Cache Memory Address
Address Translation Module 402
Address Concatenator 410
Address Translation Table Cache 412
Data Bus 408
Management Module 404
Management Registers 416
Control Sequencer 414
Management Table Cache 418
CPU Virtual or Physical Address
CPU Data
To/from Cache Control Registers

FIG. 4

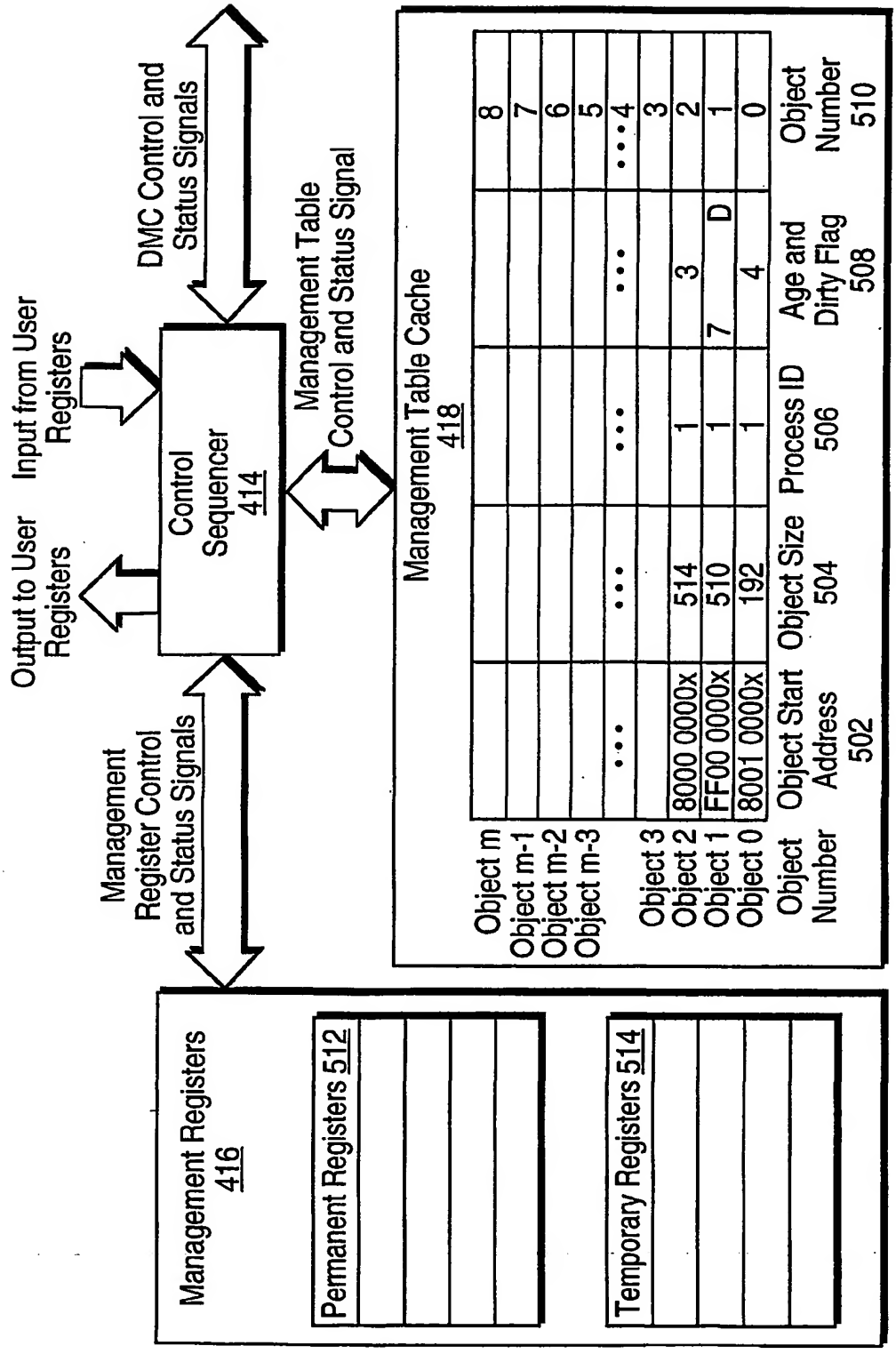
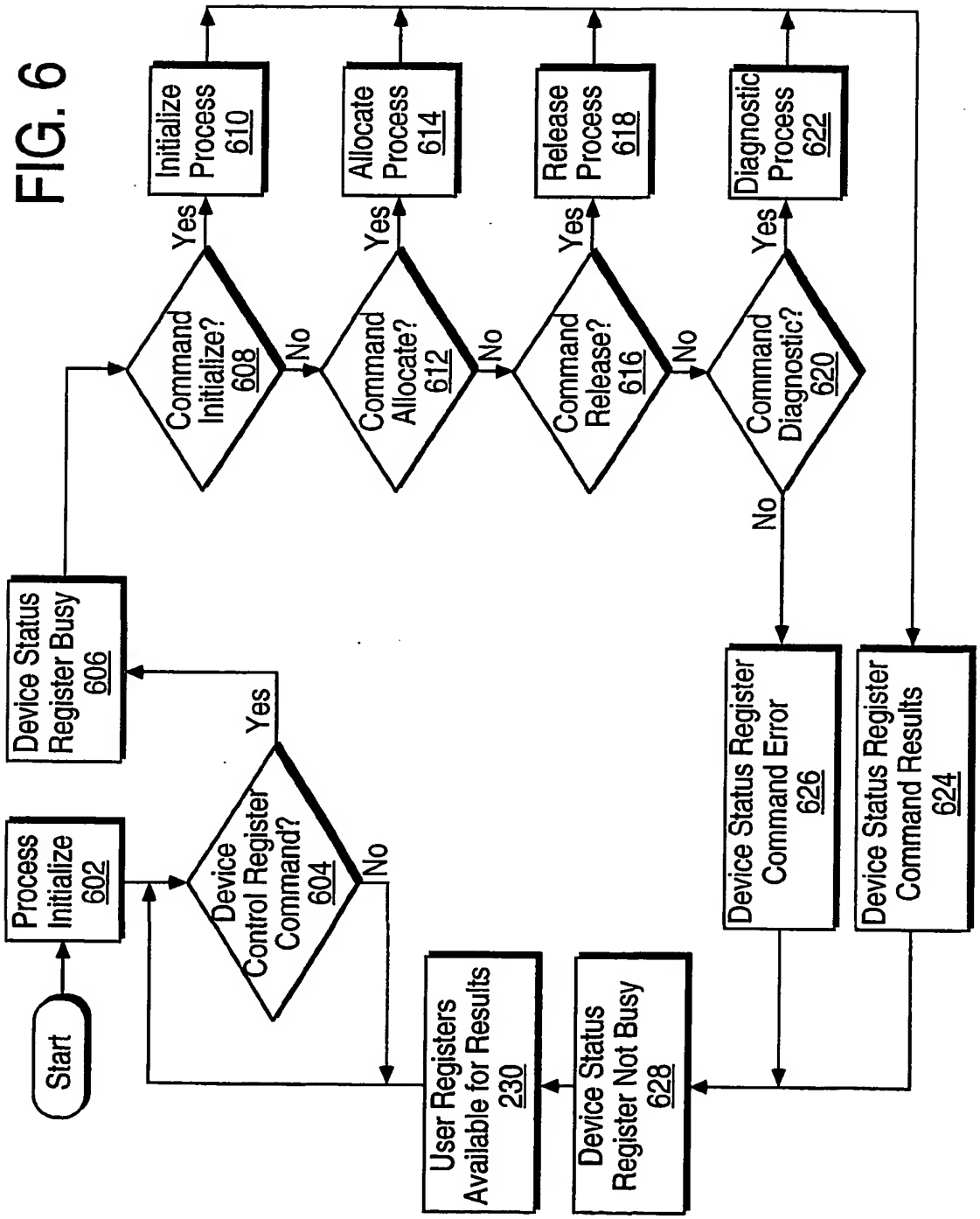

Management Registers 416
Permanent Registers 512
Temporary Registers 514
Management Register Control and Status Signals
Output to User Registers
Input from User Registers
Control Sequencer 414
DMC Control and Status Signals
Management Table Control and Status Signal
Management Table Cache 418
Object Number
Object Start Address 502
Object Size 504
Process ID 506
Age and Dirty Flag 508
Object Number 510
Object m
Object m-1
Object m-2
Object m-3
Object 3
Object 2
Object 1
Object 0
8000 0000x
514
1
3
FF00 0000x
510
1
7
D
8001 0000x
192
1
4
404

FIG. 5

FIG. 6


Start
Process Initialize 602
Device Control Register Command? 604
Yes
No
Device Status Register Busy 606
Command Initialize? 608
Yes
No
Initialize Process 610
Command Allocate? 612
Yes
No
Allocate Process 614
Command Release? 616
Yes
No
Release Process 618
Command Diagnostic? 620
Yes
No
Diagnostic Process 622
Device Status Register Command Error 626
Device Status Register Command Results 624
User Registers Available for Results 230
Device Status Register Not Busy 628

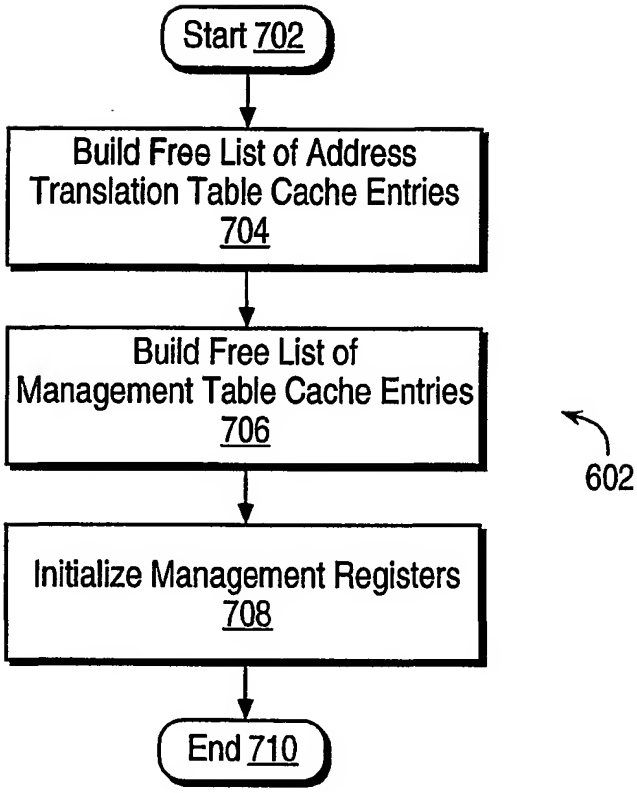

Start 702
Build Free List of Address Translation Table Cache Entries 704
Build Free List of Management Table Cache Entries 706
Initialize Management Registers 708
End 710
602

FIG. 7

FIG. 8

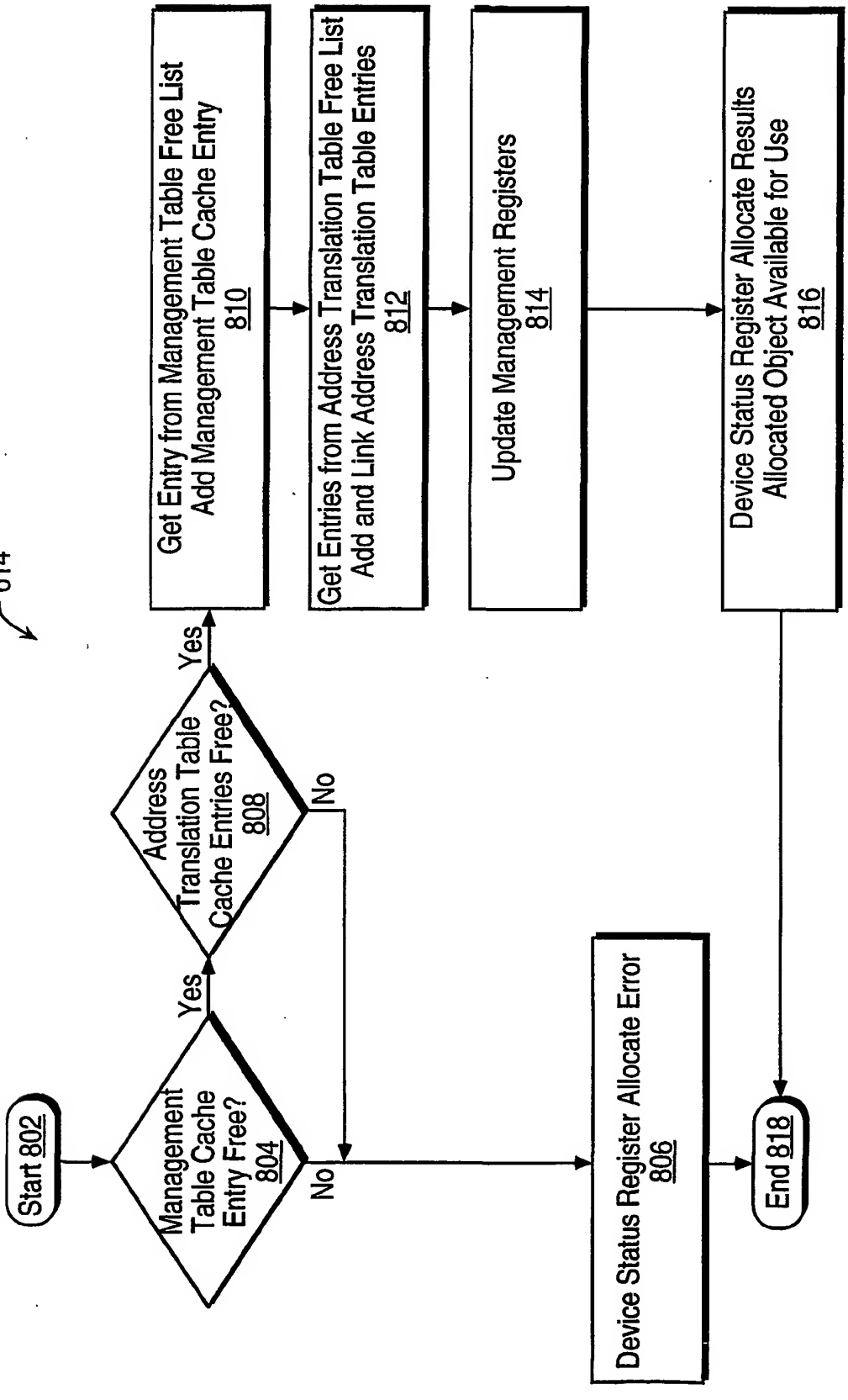

614
Start 802
Management Table Cache Entry Free? 804
Yes
No
Address Translation Table Cache Entries Free? 808
Yes
No
Get Entry from Management Table Free List Add Management Table Cache Entry 810
Get Entries from Address Translation Table Free List Add and Link Address Translation Table Entries 812
Update Management Registers 814
Device Status Register Allocate Results Allocated Object Available for Use 816
Device Status Register Allocate Error 806
End 818

FIG. 9

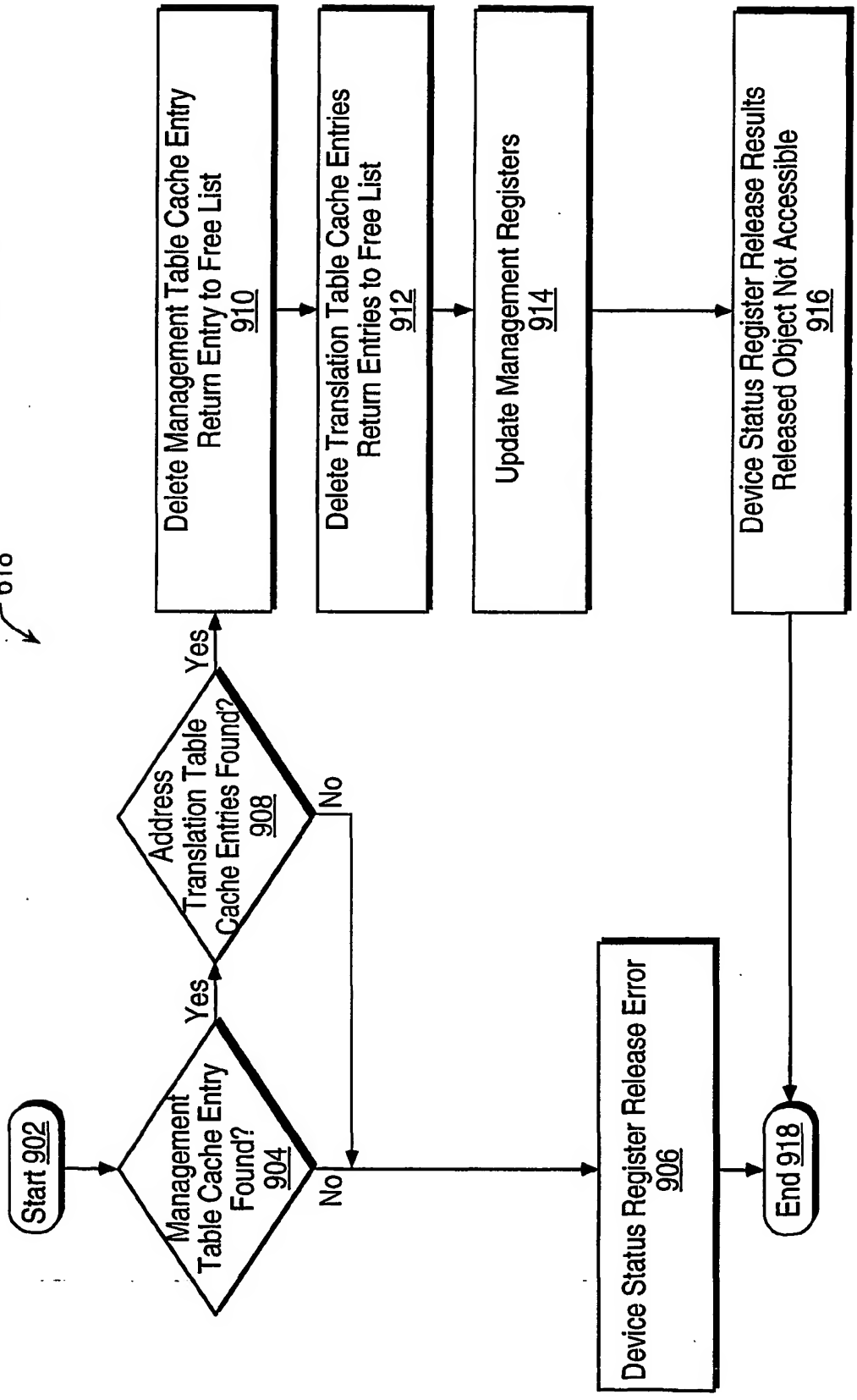
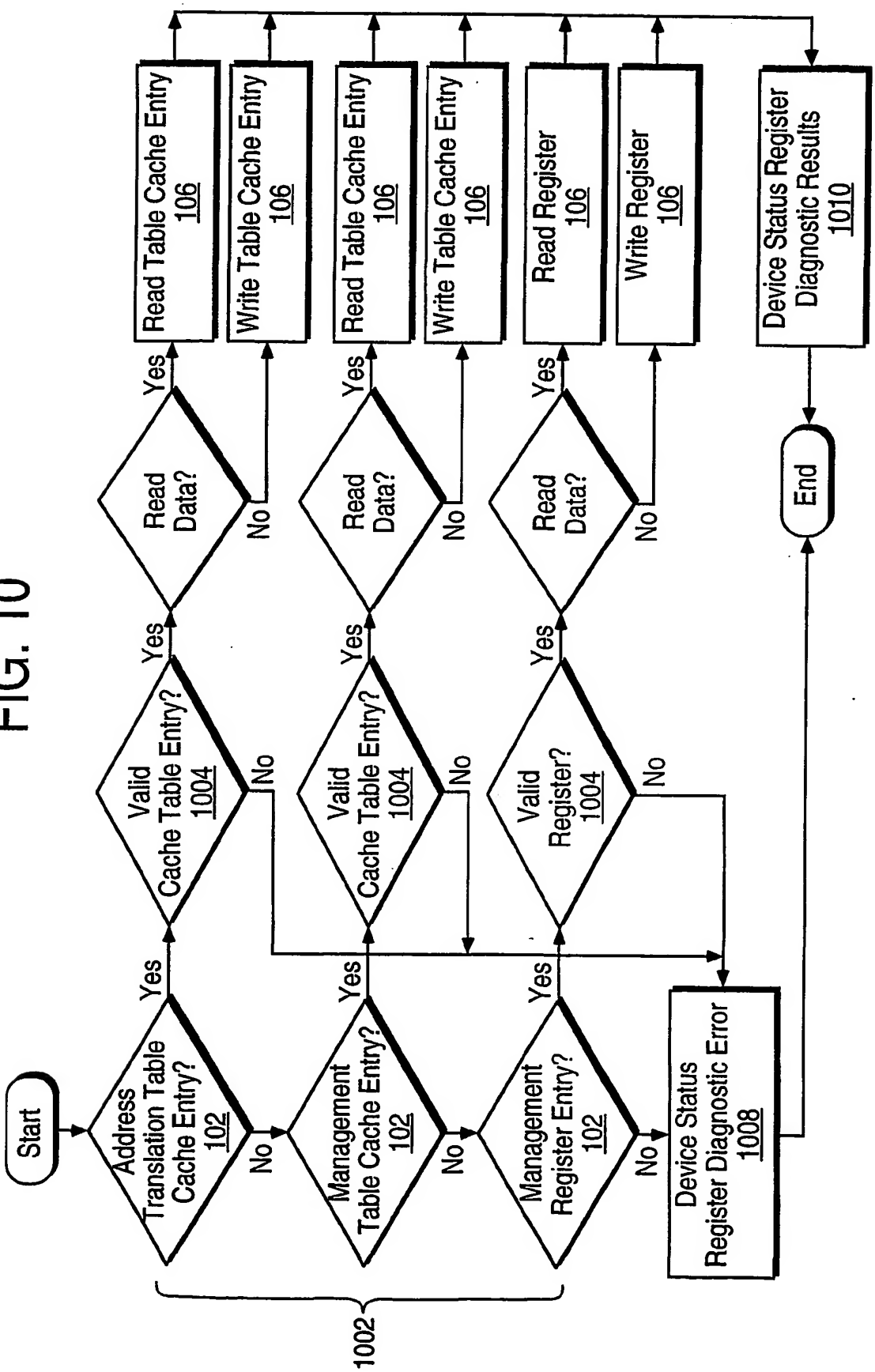

618
Start 902
Management Table Cache Entry Found? 904
Yes
No
Address Translation Table Cache Entries Found? 908
Yes
No
Delete Management Table Cache Entry Return Entry to Free List 910
Delete Translation Table Cache Entries Return Entries to Free List 912
Update Management Registers 914
Device Status Register Release Results Released Object Not Accessible 916
Device Status Register Release Error 906
End 918

FIG. 10


Start
Address Translation Table Cache Entry? 102
Management Table Cache Entry? 102
Management Register Entry? 102
1002
Yes
No
Valid Cache Table Entry? 1004
Valid Cache Table Entry? 1004
Valid Register? 1004
Read Data?
Read Data?
Read Data?
Read Table Cache Entry 106
Write Table Cache Entry 106
Read Table Cache Entry 106
Write Table Cache Entry 106
Read Register 106
Write Register 106
Device Status Register Diagnostic Error 1008
Device Status Register Diagnostic Results 1010
End

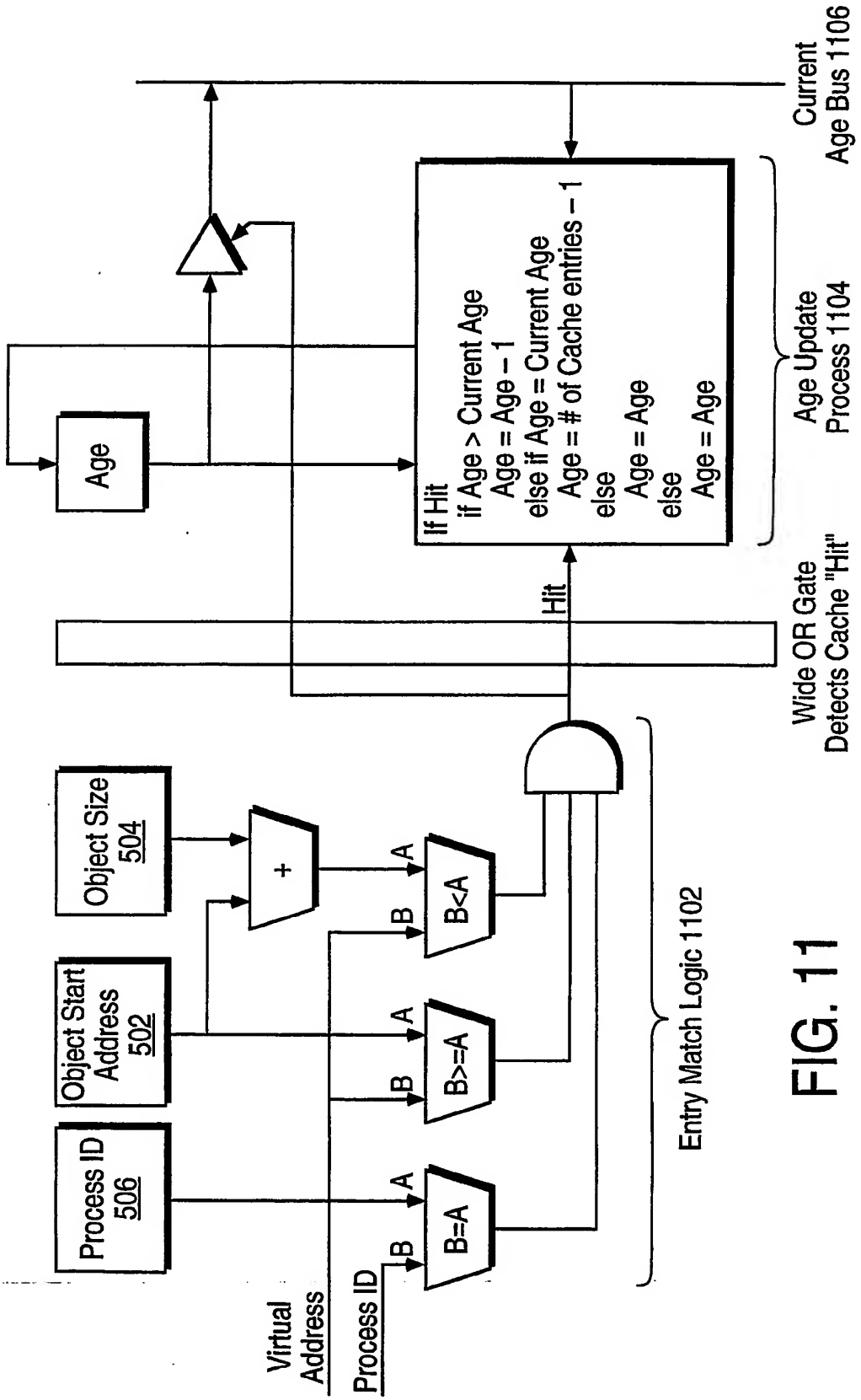

Current Age Bus 1106
Age Update Process 1104
If Hit
if Age > Current Age
Age = Age – 1
else if Age = Current Age
Age = # of Cache entries – 1
else
Age = Age
else
Age = Age
Age
Hit
Wide OR Gate Detects Cache "Hit"
Object Size 504
Object Start Address 502
Process ID 506
+
A
B
B<A
B>=A
B=A
Entry Match Logic 1102
Virtual Address
Process ID

FIG. 11

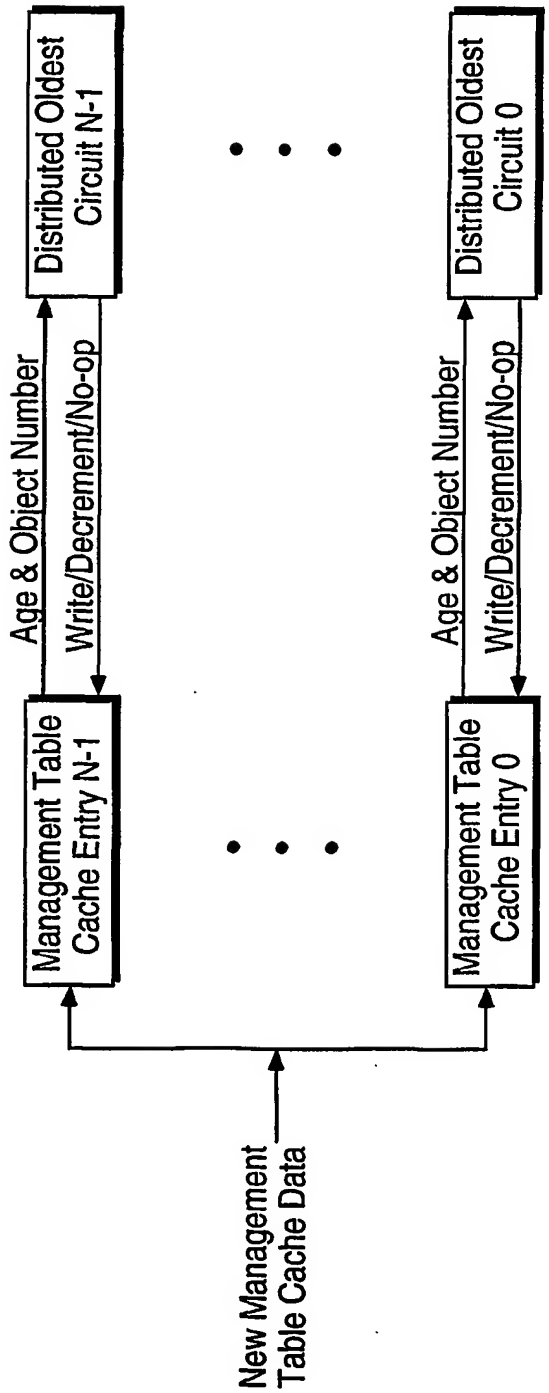

Distributed Oldest Circuit N-1
Distributed Oldest Circuit 0
Age & Object Number
Write/Decrement/No-op
Age & Object Number
Write/Decrement/No-op
Management Table Cache Entry N-1
Management Table Cache Entry 0
New Management Table Cache Data

FIG. 12

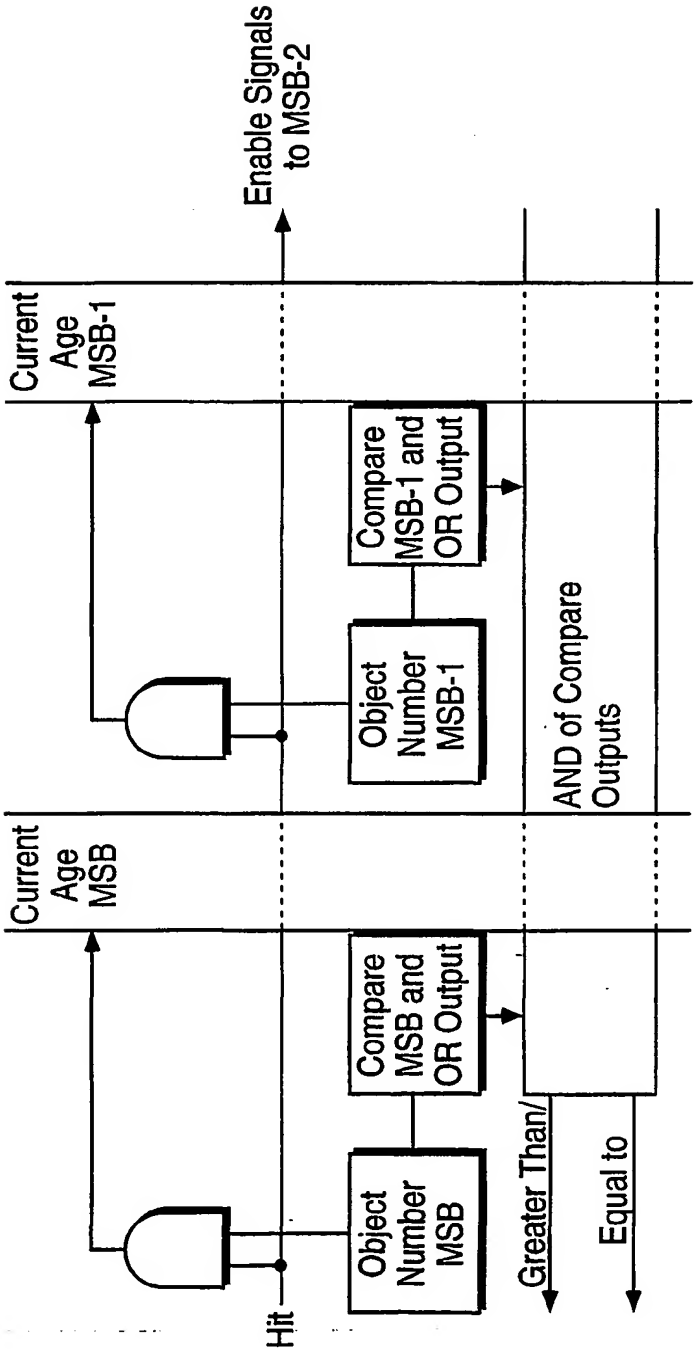

Enable Signals to MSB-2
Current Age MSB-1
Compare MSB-1 and OR Output
Object Number MSB-1
AND of Compare Outputs
Current Age MSB
Compare MSB and OR Output
Object Number MSB
Greater Than/
Equal to
Hit

FIG. 13

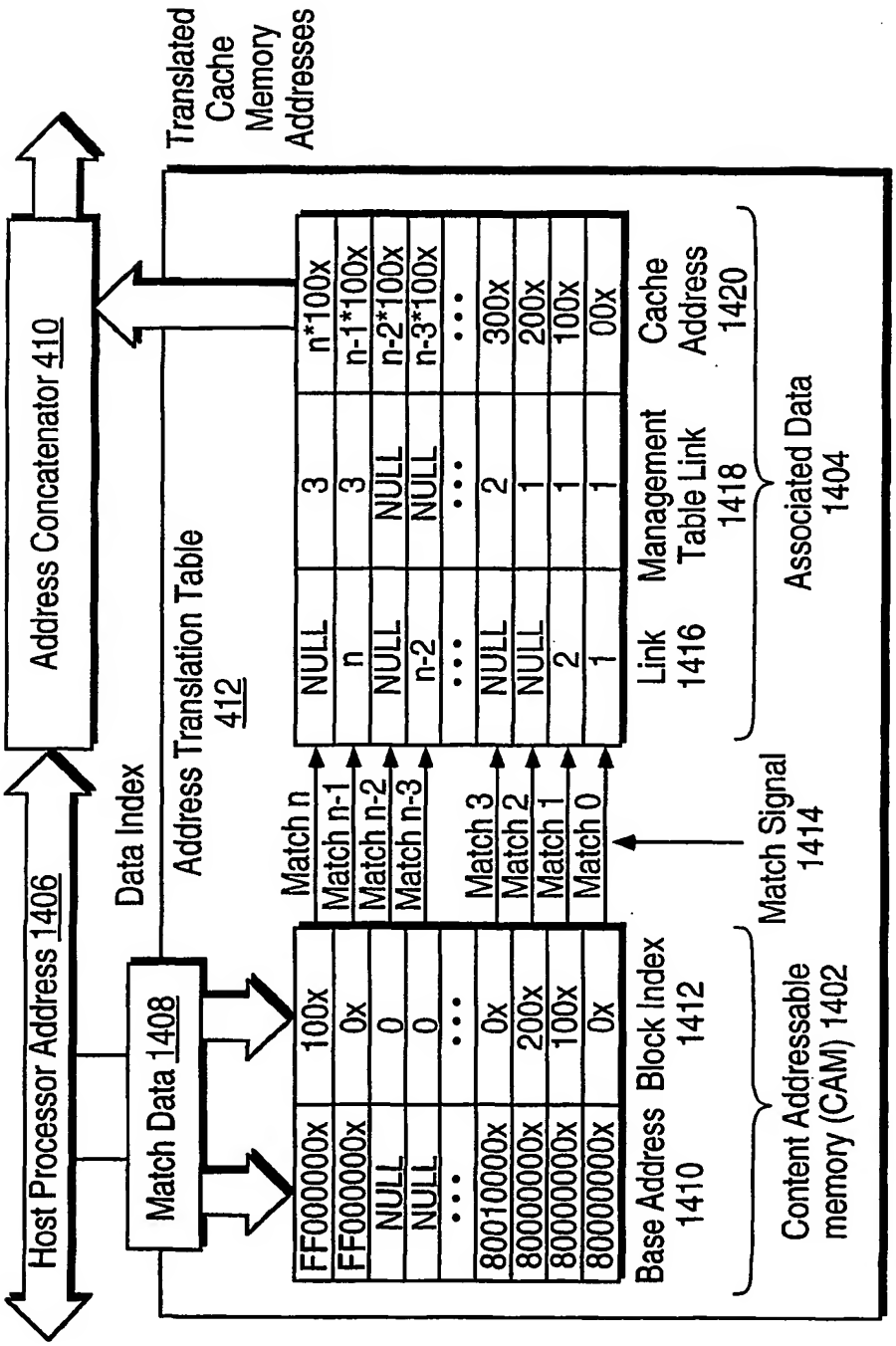

Host Processor Address 1406
Address Concatenator 410
Translated Cache Memory Addresses
Data Index
Address Translation Table 412
Match Data 1408
Match n
Match n-1
Match n-2
Match n-3
Match 3
Match 2
Match 1
Match 0
Match Signal 1414
Base Address Block Index 1410 1412
Link 1416
Management Table Link 1418
Cache Address 1420
Associated Data 1404
Content Addressable memory (CAM) 1402
402

FIG. 14

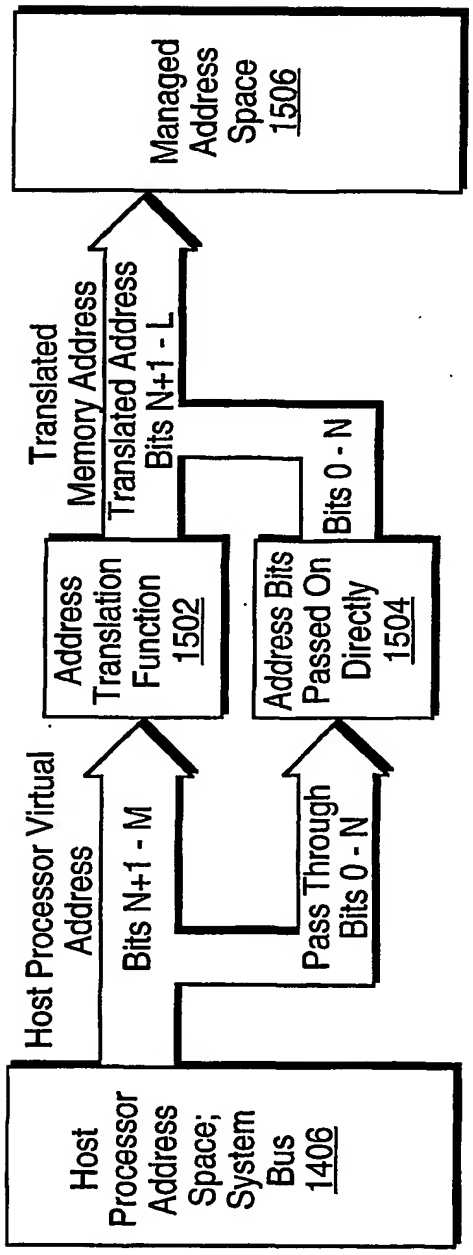

Host Processor Address Space; System Bus 1406
Host Processor Virtual Address
Bits N+1 - M
Pass Through Bits 0 - N
Address Translation Function 1502
Address Bits Passed On Directly 1504
Translated Memory Address
Translated Address Bits N+1 - L
Bits 0 - N
Managed Address Space 1506

FIG. 15

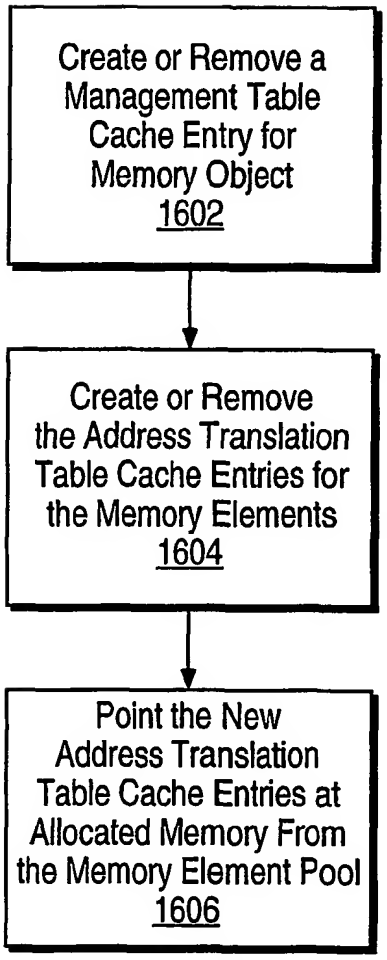

Create or Remove a Management Table Cache Entry for Memory Object 1602
Create or Remove the Address Translation Table Cache Entries for the Memory Elements 1604
Point the New Address Translation Table Cache Entries at Allocated Memory From the Memory Element Pool 1606

FIG. 16

INTERNATIONAL SEARCH REPORT

Interna al Application No
PCT/US 00/24078

A. CLASSIFICATION OF SUBJECT MAT
IPC 7 G06F12/10

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 396 614 A (KHALIDI YOUSEF A ET AL) 7 March 1995 (1995-03-07) column 7, line 52 -column 8, line 37; figure 2	1,18,24
A	EP 0 693 728 A (IBM) 24 January 1996 (1996-01-24) page 7, line 34 -page 8, line 2 page 25, line 15 -page 32, line 12; figure 6	1,18,24
A	US 5 442 766 A (CHU TAN V ET AL) 15 August 1995 (1995-08-15) abstract	1,18,24

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
30 November 2000	08/12/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Ledrut, P

INTERNATIONAL SEARCH REPORT

Interr[illegible]al Application No
PCT/US 00/24078

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5396614 A	07-03-1995	JP 7006091 A	10-01-1995
EP 0693728 A	24-01-1996	US 5729710 A	17-03-1998
		BR 9502801 A	05-08-1997
		CA 2147529 A	23-12-1995
		JP 8016412 A	19-01-1996
		KR 170565 B	30-03-1999
US 5442766 A	15-08-1995	JP 2769097 B	25-06-1998
		JP 6187152 A	08-07-1994